Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1991-09

NPSNET: an accurate low-cost technique for real-time display of transient events: vehicle collisions, explosions and terrain modifications

Osborne, William Dale.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/26651

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

NPSNET: AN ACCURATE LOW-COST TECHNIQUE
FOR REAL-TIME DISPLAY OF TRANSIENT EVENTS:
VEHICLE COLLISIONS, EXPLOSIONS AND
TERRAIN MODIFICATIONS

by

William Dale Osborne

September 1991

Thesis Co-Advisors: Dr. Michael J. Zyda
                                              David R. Pratt

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	Approved for public release; distribution is unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School	6b. OFFICE SYMBOL (if applicable) CS	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School

6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000

8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS			
	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
NPSNET: An Accurate Low-Cost Technique for Real-Time Display of Transient Events

12. PERSONAL AUTHOR(S)
Osborne, William Dale

13a. TYPE OF REPORT Master's Thesis	13b. TIME COVERED FROM 08/89 TO 09/91	14. DATE OF REPORT (Year, Month, Day) September 1991	15. PAGE COUNT 42

16. SUPPLEMENTARY NOTATION   The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17.	COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Collision Detection, Real-Time, Ray Tracing, NPSNET, Computer Graphics.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This work concentrates on a method for real-time collision detection and how to resolve that collision when it has occurred. The results of this effort are only a small part of the overall system, NPSNET. The collision detection mechanism is integrated into the overall system to create realism involving collisions. The original NPSNET system did not contain a collision detection and response module. The collisions to be detected include explosions such as missile contact with a vehicle, one vehicle running into another such as a jeep and a tank, and terrain modifications such as an artillery round hitting the ground and creating a crater. The overall system complements the DoD large-scale networking system, SIMNET. The NPSNET system is portable and able to run on any graphics workstation that has the GL libraries.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Michael Zyda	22b. TELEPHONE (Include Area Code) (408) 646-2035	22c. OFFICE SYMBOL CS/Zk

NPSNET: An Accurate Low-Cost Technique
for Real-Time Display of Transient Events: Vehicle Collisions,
Explosions and Terrain Modifications.

by
William Dale Osborne
Captain, United States Army
B.S., United States Military Academy, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1991

ABSTRACT

This work concentrates on a method for real-time collision detection and how to resolve that collision when it has occurred. The results of this effort are only a small part of the overall system, NPSNET. The collision detection mechanism is integrated into the overall system to create realism involving collisions. The original NPSNET system did not contain a collision detection and response module. The collisions to be detected include explosions such as missile contact with a vehicle, one vehicle running into another such as a jeep and a tank, and terrain modifications such as an artillery round hitting the ground and creating a crater. The overall system complements the DoD large-scale networking system, SIMNET. The NPSNET system is portable and able to run on any graphics workstation that has the GL libraries.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

This work covers the collision detection and response portion of a new battlefield simulation package, NPSNET [Zyda91]. NPSNET is a commercial workstation-based version of the older, more expensive system, SIMNET [Garv88]. NPSNET is programmed utilizing off-the-shelf IRIS graphics workstations rather than the platform specific nodes of SIMNET. This new system provides a real-time interface with the user in the physically-based world model.

A. BACKGROUND

NPSNET is a real-time vehicle and battlefield simulator. It uses databases and formats similar to those found in SIMNET, the Department of Defense (DoD) large-scale networking simulator. Users of NPSNET are able to drive many different types of vehicles such as tanks, jets, ships, helicopters and armored personnel carriers. There are also a number of humorous objects that are available for the more adventurous, such as helo-cows, wishbones, hamburgers and even attack tomatoes. Up to 500 of the vehicles/objects can be driving in the world at one time. The 500 vehicles can include autonomous vehicles that react when fired upon by either returning fire or fleeing the area. These three dimensional (3D) icons move around in the computer world that is based on the terrain at Fort Hunter-Liggett, California. There are trees, bushes, rocks, roads, watertowers, buildings and much more that are on the rolling hills and valleys for the virtual Fort Hunter-Liggett. Depending upon the model of IRIS being used, the user can select to use texturing, fog and even haze. The system is networked, via Ethernet, to allow several players to interact. A two dimensional (2D) map can be displayed that shows the position and tracking of all the players in the 50 kilometer square. This map displays the direction and viewing

1

triangle of the driven vehicle as well as the position and movement of the remaining vehicles. The statistics and data concerning the driven vehicle are displayed in a window at the top of the screen. Speed, pitch, roll, number of remaining rounds and remaining fuel are a few of the statistics shown. Players control their chosen vehicles through several interface devices to include a button/dialbox, keyboard and SpaceBall.

The six degree of freedom SpaceBall is one of the most versatile devices available and allows for control of movement in 3D. The pick button on the SpaceBall fires the appropriate round associated with the vehicle being driven. Pressure applied to the SpaceBall adds to the thrust in the applied direction. The more thrust applied the faster the change. This allows the player to turn, move forward, backward, up and down very quickly. The system runs in real-time. Consequently, reactions to events, not just the detection of them, have to take place very quickly also. This includes following terrain contours, reacting to input from the user and responding to changes in the 3D world. As long as no collision occurs, the displays on the original NPSNET, NPSNET-1, are realistic.

B. PURPOSE AND GOALS OF WORK

NPSNET-1 has no collision detection, so collision response is not done. Without collision detection and response the realism of NPSNET-1 is poor. Even with texturing, environmental effects and realistic looking vehicles, the virtual world falls apart the first time one vehicle drives through another. This work places collision detection and response into NPSNET-2. Many of the scenarios in NPSNET-1 are not realistic since a user can drive through walls, trees, other vehicles and any object that is encountered. This work, implemented in NPSNET-2, detects and responds to collisions between objects in real-time. This is difficult due to the computational intensity of collision detection. Speed is vital to detection as well as a realistic response. The detection portion is fast to allow the time needed to respond properly. Response is dependent upon physically-based modeling.

Physically-based modeling is the process of giving objects the characteristics they actually possess and making those objects react to the forces that influence them in the real

2

world. Transient events are those inputs that act upon the models to change the characteristics of the models. For example a missile impacting upon a tank would definitely change the tank's representation. Characteristics include things such as: spring forces, moldability, rigidity, weight, gravity, explosive potential and much more. Transient events include collisions, explosions, terrain modifications and anything else that affects the physically-based model of either the world itself or the individual objects within that world. There are physically-based models on the market with realistic texturing, collision detection and response. However, very few are done in real-time.

Achieving real-time collision detection and response is the primary goal of this work. Secondary goals include compatibility with SIMNET, realism in the responses and compatibility with future hardware upgrades. Detection of all collisions is another secondary goal. This goal may not be necessary though since realism can be achieved by only responding to those collisions that occur within the viewing area of the user.

C. BREAKDOWN OF WORK

Chapter II covers previous attempts to solve the problem of collision detection and response including the one it complements, SIMNET. Other solutions to this problem are covered including interpenetration of spheres and boundary boxes, physically-based modeling and simple spring forces.

Chapter III covers a description of the program. Several steps have been taken to limit the number of objects checked for a collision. These are also discussed in Chapter III. The implementation of the program is discussed along with the various algorithms and thought processes that went into its design. Efficiency and data structures are cited along with various examples of the code itself.

Chapter IV covers the results of this work. Chapter V covers the conclusions. This chapter lists requirements and suggestions for future work in the area of real-time collision detection and response for NPSNET-2.

II. OTHER RELATED WORKS

This work covers collision detection and response in real-time. There are several papers that cover collision detection and response but not in real-time: [Moor88], [Hopc83] and [Terz87]. There are also a few works that cover it in real-time: [Garv88], [Hahn88] and [Uchi83]. However, none of these works use a commercial workstation-based system for hardware.

A system for an interactive battlefield simulation is SIMNET [Garv88]; however, it is prohibitively expensive and only runs on a particular set of hardware. Additionally, SIMNET requires specific hardware for each type of vehicle whereas NPSNET uses one general purpose simulator for all vehicles. SIMNET has a collision detection and response system which is only a small part of the overall system just as this work is only a small part of the NPSNET system. NPSNET complements SIMNET as a general battlefield simulation system; however, there are other existing simulation and collision detection papers which approach the problem of collision detection and response specifically.

An example is a paper by Moore and Wilhelms [Moor88]. It discusses the issue of collision detection and response very specifically and goes into detail about both flexible and solid surfaces. The algorithm presented in this paper tests to see if the points of one object are inside the points of another, and if they are, a collision has occurred. Two algorithms for collision detection are given in Moore and Wilhelms' paper, each of which is broken down into two parts. One part tests for planar penetration, and the other part tests for edge penetration. The results of both algorithms are then passed to a collision response algorithm. The algorithm then determines an appropriate response to the collision. The response concentrates on giving new linear and angular velocities to the objects involved. The authors take two approaches: one for objects at rest with forces acting upon them, such

4

as gravity and mass and a second approach for moving objects. The at rest objects respond with spring-like reactions while the moving objects have to be analyzed to determine the appropriate response. Physically-based modeling is discussed and partially implemented. The paper's final solution to the problem of collision response is to use a dynamic approach which can access either the spring force or the analytical method. The biggest drawback to the paper is that the implementation is not done in real-time.

Another paper is *Collision Detection in Motion Simulation* by Uchiki, Ohashi and Tokoro [Uchi83]. It uses an independent process, a space occupancy method, which detects when spheres, which enclose objects, occupy the same space. Each object is sent a message whenever it tries to occupy a space that is already occupied by another object's sphere. Consequently, message passing is the key to its success. It has an additional feature that makes it unique in that it also passes the point of the collision to the collision detector. This is an important bit of information that is essential to collision response. To properly react to a transient event, the collision point must be known. For example, the system should crumple on the right side if the right side is hit. Again this is a characteristic of physically-based modeling and one that must be preserved in order to accurately and realistically display interaction among objects in the physically-based world.

The paper by Hopcroft, Schwartz and Sharir [Hopc83] provides an algorithm for determining whether a collision has occurred between two objects in three dimensional space. The paper uses spheres to determine intersections between objects. Every object within the paper's model is enclosed within a sphere. The basis of the paper is to determine if any two of those spheres intersect. It also provides a computational complexity analysis for the algorithm along with the mathematics involved in calculating the intersection. The entire approach of the paper is mathematical in nature. Hopcroft's paper also contains the data structures used to create the efficiency of their method. Sorting and placement of the sphere locations and radii are an important part of the method and contribute greatly to the results obtained.

In Hahn's paper [Hahn88] an overview and a limited implementation for a computer animation system to model 3D moving objects is presented. Hahn's paper goes into detail on the physically-based modeling of the objects and the methodology for creating realistic movement of those objects. The paper provides a method for computing the motion of objects by merging not only dynamics but kinematics as well. It allows for interaction between objects that includes collision detection and response. If a collision has occurred, then the collision point along with the backup vector is sent to an analyzer which determines the appropriate response. The response is limited to a bouncing effect at a new velocity and angle. This is the major shortfall of the paper and where the physically-based modeling stops. Responses are built into a table of script files and are limited so that a true response may not be given but whatever comes closest to matching the pre-programmed response.

In *Elastically Deformable Models* by Terzopoulos, et al [Terz87] the problem of deformable objects is discussed. These are objects which would not normally get penetrated but would respond by giving in or bending away from the collision point. Objects of this type include things like paper, rubber and other flexible materials. The paper deals exclusively with deformable objects just as its title states. *Elastically Deformable Models* does demonstrate what occurs in response to different types of forces, constraints and other objects. Their paper promotes the use of dynamic models which react to transient events based upon the principles of applied physics.

A predecessor to NPSNET is the moving platform simulator (MPS) series. It has three versions, and versions two (MPSII) [Winn89] and three (MPSIII) [Chee90] each have collision detection. The detection consists of a 2D check for nearness of other vehicles or platforms. If a platform comes within a certain range of another platform then both platforms are killed. There is no check for non-platforms, such as trees, bushes, etc. Additionally, the only response is to kill the vehicles, not damage them or bounce them off of each other.

The fundamentals of detecting collision points are contained in the book *An Introduction to Ray Tracing* [Glas89]. This book covers not only the fundamentals but the specifics of finding intersection points for collisions. Finding the intersection points is discussed in detail in Chapter III of this work.

Although, several other programs and systems exist that perform collision detection and response, few do so on SGI IRIS graphics workstation hardware and none do it in real-time. SIMNET comes closest to meeting these objectives but works only on its own particular set of hardware.

III. PROGRAM IMPLEMENTATION

A. OVERVIEW

NPSNET-2 runs on any graphics workstation with the GL libraries but has been developed on the IRIS workstations, including the IRIS 4D/120 GTX, 4D/70 GT and the 4D/240 VGX. It is written in Kernighan & Ritchie C [Kern78]. Input and output devices that are supported include the keyboard, button/dialbox, mouse, screen and SpaceBall. The NPSNET system involves real-time response in a battlefield simulation using land, sea and air forces. The system is networked, via Ethernet, to allow for multiple players to interact. This work performs realistic animation of explosions involving direct and indirect hits by ordnance; collisions of vehicles with other vehicles and terrain features; and terrain modifications such as craters and destroyed trees. NPSNET is relatively inexpensive in comparison to SIMNET. Moreover, NPSNET uses one general purpose simulator to operate on the entire battlefield while SIMNET uses a different type of simulator for each different type of vehicle/platform. This allows any user to sit down at one terminal and become any vehicle in the simulated world that he wants to. If the user changes his mind at any time about his choice of vehicles, he can change by simply pressing a button rather than switching hardware. Due to the generic application of the collision detection routines, NPSNET-2 continues to perform in real-time regardless of the simulated vehicle.

B. WORLD SEGMENTATION

NPSNET's virtual world is divided up into gridsquares of a constant size based upon the actual terrain features of the database from the SIMNET Database Interchange Specification (SDIS) [Lang90]. The gridsquares are small, 125 meters square. Associated with each gridsquare are both fixed and moving objects.

8

C. COLLISION DETECTION

In order to obtain a realistic virtual world there must be collision detection. Vehicles passing through other vehicles and objects make the world unrealistic. A possible solution to this problem would be to prevent collisions by bouncing objects off of each other at all times, but that is not very realistic either. Another possible solution is to always destroy the objects involved in collisions. A third option is to combine these two solutions along with varying stages of damage to involved objects depending upon the physical characteristics of the involved objects. That is the approach taken by this work.

1. Against Fixed Objects

The algorithm for collisions with fixed objects constantly checks moving vehicles to determine if a collision has occurred. The position of the moving vehicle is updated constantly. Consequently, as soon as a vehicle is moved and its position is updated, it is checked for a collision. In order to maintain a real-time speed, the scope of the collision detection is severely limited. This is done in two basic ways. The first is a collision with fixed objects is checked only if the moving vehicle is below a threshold height. This is due to the fact that all fixed objects are in some way attached to the terrain and thus below that threshold height. If it is below that height, it runs through a linked list of fixed objects which are attached to the current gridsquare. This is a quick check since there are relatively few fixed objects in any one gridsquare (Table 1).

Associated with each object in the linked list is a radius that is used for its bounding sphere. If that bounding sphere is interpenetrated by the bounding sphere of a moving object, then a collision occurs. Depending upon the mass of the fixed object, there are various outcomes to the collision. A large massed object is much less vulnerable to damage than a small massed object. Consequently, the larger the mass of the fixed object, the less damage it suffers, and the more damage the moving object suffers. The opposite is also true. If there is no collision between the fixed objects in the gridsquare and the moving object, the second collision check is performed.

Minimum per Gridsquare	0
Maximum per Gridsquare	18
Average per Gridsquare	3.9

OBJECT TYPES	TOTAL
Trees	30,813
Bushes	9,422
Telephone Poles	54
Buildings	3
Watertowers	1
Other Man-made	5
Other Natural	5
Total for Database	40,303
Total 125 Meter Gridsquares	160,000

DISPERSION OF FIXED OBJECTS
TABLE 1

2. Against Moving Objects

The second collision check is for other moving objects. This is more complicated since any other moving vehicle or object has the potential for colliding with the vehicle we are checking. The potential exists for 500 vehicles including their missiles and other ordnance to be checked for collision. Consequently, the scope of the collision detection range has been limited in several ways.

As soon as each vehicle is moved, its position is checked against the position of the surrounding vehicles. If the X or Z position of any other vehicle is within 100 meters of the checked vehicle then those two vehicles are sent to the second level check. At the second level check the distance between the two vehicles is calculated. If this distance is less than the combined radii of the two vehicles then a collision has occurred and the third level collision check is done. Ray tracing, the third level check, determines the actual point of collision if there is one.

If worst case numbers are used to determine the implicit range limitations of all vehicles, it can be shown why this culling is fairly accurate. Reasonable speed limitations of the various types of vehicles are used to calculate worst cases for each (Table 2). For

	KPH	Meters/Sec	Frames/Sec	Meters/Frame
Ground	60	16.6	10	1.66
Ship	50	13.8	10	1.38
Air	1000	277.7	10	27.7

VEHICLE MOVEMENT LIMITATIONS
TABLE 2

example, if a ground vehicle travels at 60 kilometers per hour, then it travels 1000 meters per minute or 16.6 meters per second. At a frame rate of 10 frames per second, this is equivalent to 1.66 meters per frame. Since the vehicle positions are updated each time before the frame is displayed, they are also checked for a collision. A ground vehicle would have to travel at approximately 1000 KPH to completely traverse two gridsquares in one second. Consequently, the movement across more than two gridsquares within one tenth of a second, one frame, is impossible (Figure 1). The distance for the first level check is used
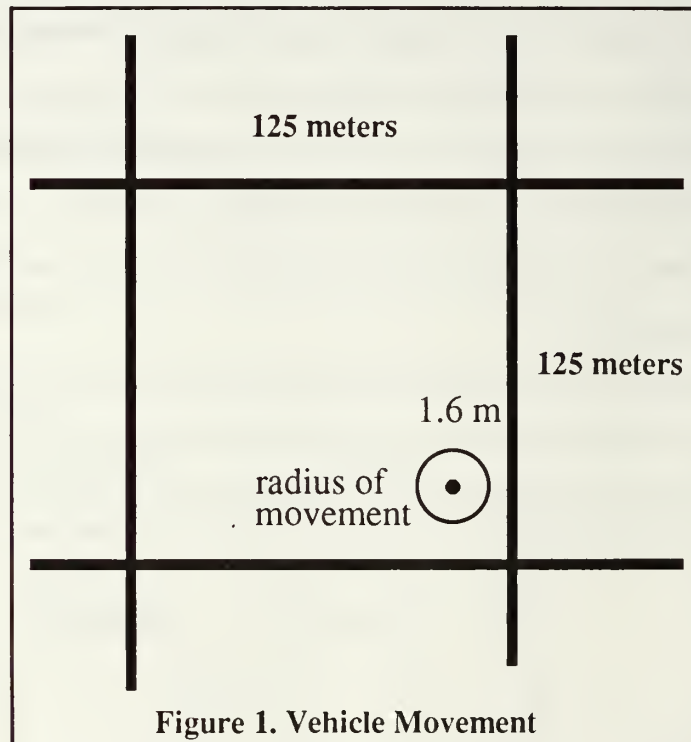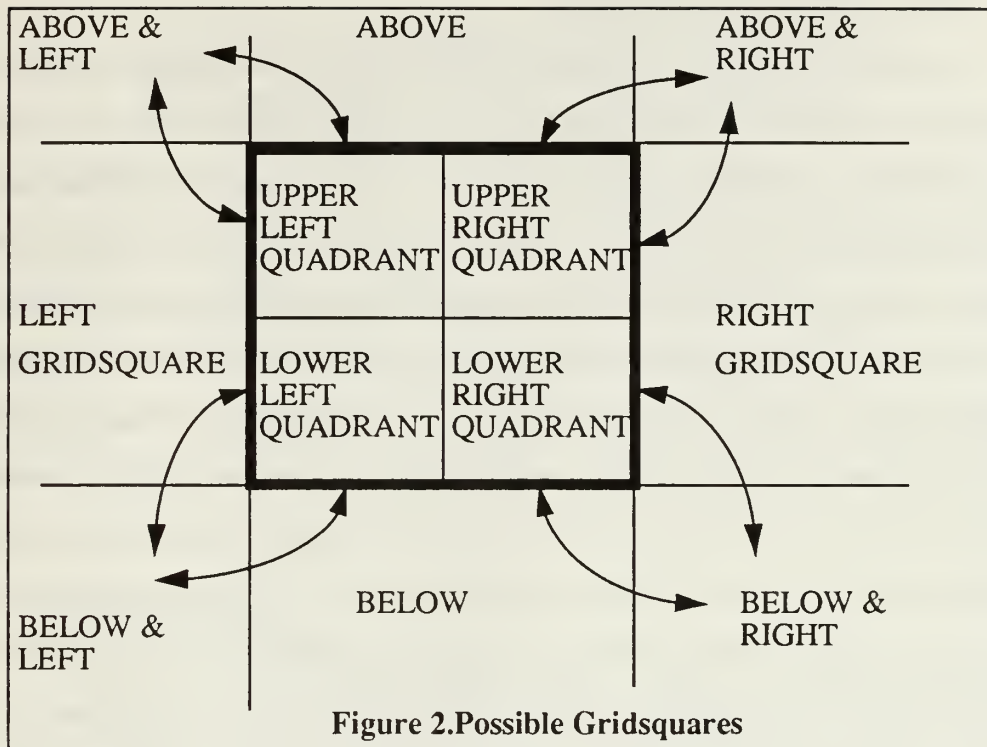

Figure 1. Vehicle Movement

as a rough approximation for proximity of other vehicles. The only gridsquares that can be reached by the vehicle within one frame are those that are checked as shown in Figure 2. The limitation of the 100 meters ensures an efficient culling for collision detection and allows the time needed for collision response.

The collision detection itself is done by determining if one object has interpenetrated another. If an interpenetration has in fact occurred, then it must be resolved. The most obvious way to determine if a collision has occurred is to create a boundary box around each object and if that box gets penetrated, then a reaction must occur. The

ABOVE & LEFT	ABOVE	ABOVE & RIGHT
	UPPER LEFT QUADRANT	UPPER RIGHT QUADRANT	
LEFT GRIDSQUARE	LOWER LEFT QUADRANT	LOWER RIGHT QUADRANT	RIGHT GRIDSQUARE
BELOW & LEFT	BELOW	BELOW & RIGHT

Figure 2.Possible Gridsquares

penetration boundary can also be done by surrounding each object with a sphere. Both methods are simple to implement, but the sphere implementation is slightly faster. Therefore, bounding spheres serve as the outer bounds of the objects in this work.

Neither method calculates the penetration point. The radius used in the spherical check is the maximum distance from the center of the object to the furthest outer surface. The boundary box uses a maximum and minimum value, not necessarily the value at that part of the object being penetrated. Consequently, in the collision response portion of the system, the actual object's penetration point is determined. A slightly smaller value than the actual radius of the object is used for the radius. This produces a more realistic collision possibility since it increases the likelihood of an actual collision of the checked objects and not just their spheres. Once the collision has been detected, the function to determine the extent of damage and the results to be displayed is run.

D. COLLISION RESPONSE

Collision response is handled by a function which takes in the two involved objects as arguments and determines the impact of the damage upon the them. For example two tanks colliding, each going five miles per hour, would have a much smaller impact than a tank going 40 miles per hour hitting a stopped jeep. An artillery round striking the ground leaves a large crater while the same round striking a tank would destroy the tank and have no effect on the ground. Many variables must be taken into account to include speed and angle of impact, mass of the objects involved, explosive potential, resistance to destruction, moldability of the objects, rigidity and fabricated spring forces which determine the bouncing-off effect and likelihood of survivability. Each of these factors is weighted in order to provide as realistic an effect as possible while maintaining the environment in real-time. For example, if a tank runs directly over a tree quickly, there should only be a stump remaining if the vehicle operator were to turn around the tank and look back to where he had just driven from. Additionally, if two tanks were to collide at 20 miles per hour, there would probably be a large dent in both along with a severe bouncing effect if the angle of impact was small. If the angle of impact was severe, then both tanks would sustain a large amount of damage. In a real situation, there would be several visual effects that would occur simultaneously in response to the impact. Special effects such as smoke and fire are included.

1. Fixed Objects

The implementation of collision response requires the input of the two objects involved. A basic assumption that was made was that collisions between more than two objects do not occur very often. Therefore, the collisions checks and responses in this system only involve two objects. It was felt that this was a valid and justified assumption. Associated with each object, both fixed and moving, are radii that determine the sphere size for the collision checks. For a moving vehicle colliding with a fixed object, there are only a few basic cases:

1. the vehicle is undamaged and destroys the fixed object;
2. the fixed object is damaged and the vehicle is also or
3. neither the fixed object or the vehicle are damaged.

If the fixed object is large and heavy, like a building, then the vehicle is probably going to be destroyed. Whereas if the fixed object is small and light, like a small tree or stop sign, the fixed object will be destroyed. If a jeep runs over a bush, neither one of the objects will be damaged. There are a few cases where both the vehicle and the fixed object will be damaged such as a tank hitting a house. Most of the damage would be to the house, but the tank would suffer in the collision also. A more complex issue arises when two moving objects impact with each other.
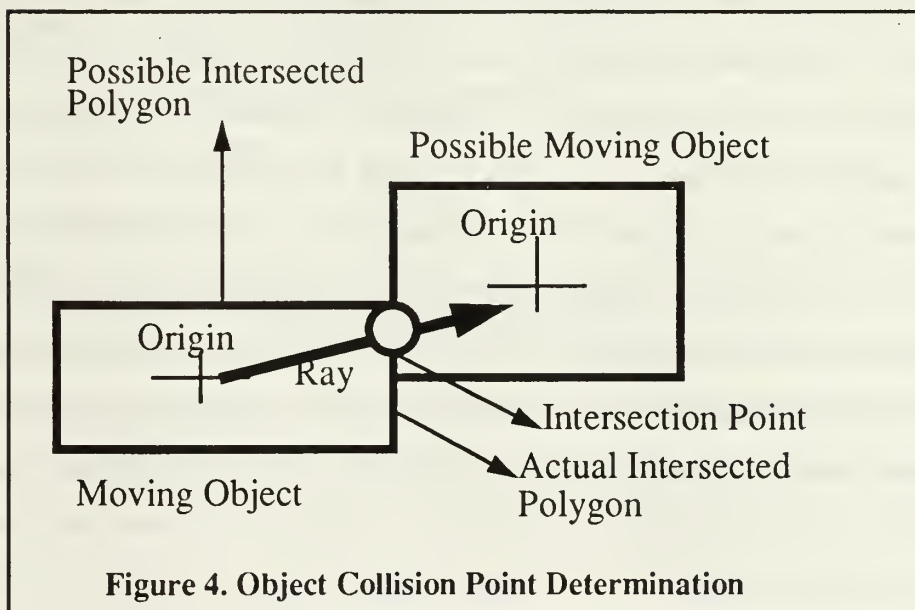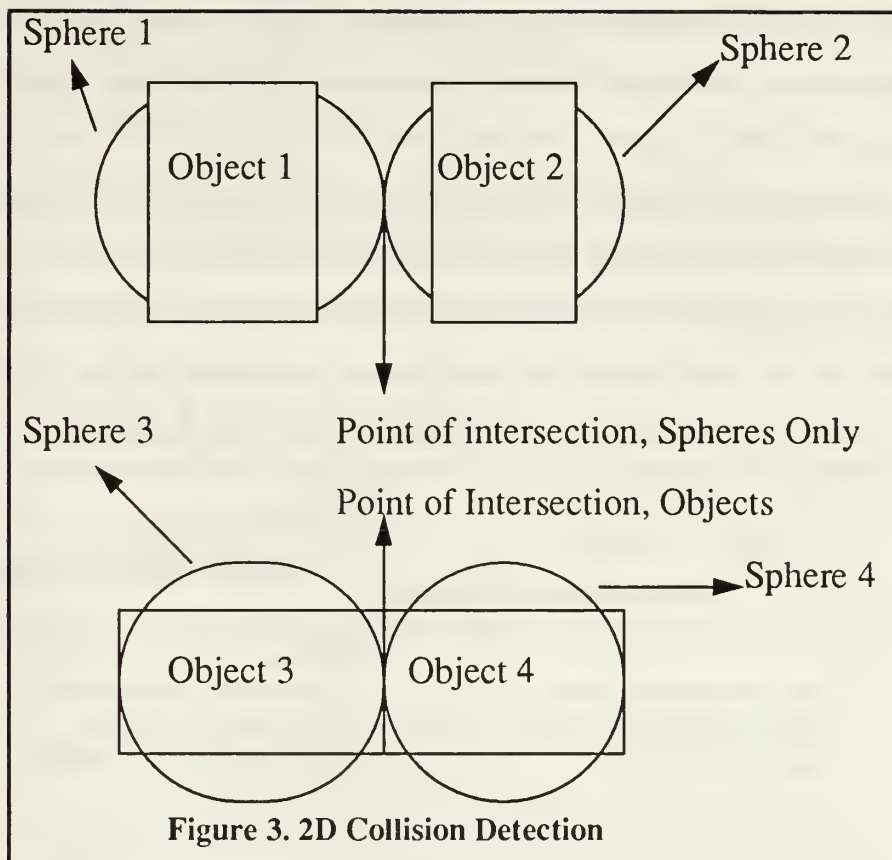
2. Moving Objects

In the case where two moving objects impact, all of the physically-based modeling characteristics of each object must be considered. The collision point in three dimensions must be known to create realistic responses in the involved objects. The collision point determines the point for any type of bending, crumpling and molding. Moreover, if the point of collision is part of a wall that is interconnected to several other walls then there will have to be corresponding responses in those interconnected walls. The only way to find the collision point is through ray tracing. The ray does not bounce around forever but only long enough to give the x, y and z coordinates of the first collision point, if one exists. Normally, ray tracing is expensive computationally. However, only the first intersection point for each ray is computed; therefore, the expense is lessened considerably. A form of backward ray tracing is used with the origin of the first ray being the origin of the object itself and the origin of the second ray being the possible intersection point. Two rays are needed for different portions of the program.
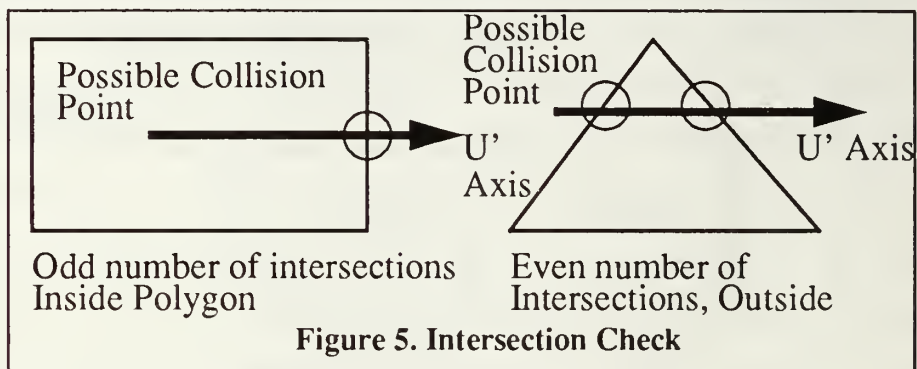
A ray shot from the center of the moving object towards the center of the object it collided with determines the possible point of collision. This is done since the collision detection portion returned a true for the collision along with the two objects involved. This

collision may simply be between the spheres of the two objects though and not the actual objects themselves. This possible intersection point, produced from the first ray, is where a collision would occur on the surface of the sphere used for the collision detection part of the algorithm. Due to the possibility that the actual object was not penetrated, the possible collision point is used as the origin of the second ray.

The second ray determines if one of the object's actual polygons was penetrated. This second ray is the ray used in Haines algorithm. This algorithm from Glassner [Glas89] was adapted for use in the collision point determination. It involves running through the list of polygons that comprise the object. Each polygon is checked until an intersection is found. Since this portion of the program is only done if there is a collision between the spheres, the ray is shot through the planes that make up the object's polygons until the actual intersected polygon is found. If no intersection is found once all of the polygons have been checked, then only the spheres were penetrated and not the objects themselves. Figure 3 shows this in 2D. So although several planes which extend indefinitely from the plane of the polygons could be intersected, only one polygon will be intersected. In two dimensions, if we imagine that the object is a box as in Figure 4, then the front face is intersected. However, if the plane for the upper side were extended indefinitely then the side plane would be intersected also. When the actual polygon on the upper side is checked no intersection is found. Consequently, the algorithm would have to be run again until an actual polygon intersection were found in the front face. Figure 4 shows a 2D representation of this.

Figure 3. 2D Collision Detection

Labels in figure: Sphere 1, Sphere 2, Object 1, Object 2, Point of intersection, Spheres Only, Point of Intersection, Objects, Sphere 3, Sphere 4, Object 3, Object 4

Figure 4. Object Collision Point Determination

Labels in figure: Possible Intersected Polygon, Possible Moving Object, Origin, Origin, Ray, Intersection Point, Actual Intersected Polygon, Moving Object

If the plane that the polygon lies in is intersected then the polygon itself must be checked for an intersection. This is where Haines' adaptation of the Jordan Curve Theorem is implemented. The Jordan Curve Theorem simply states that if a point lies inside a polygon and a line is drawn from that point to the outside of the polygon then it will intersect the polygon edges an odd number of times. Conversely, if the polygon edges are intersected an even number of times by the ray from that point, then that point lies outside the polygon. Therefore, if the polygon is intersected an odd number of times by the second ray shot from the saved intersection point then the intersection point lies within the polygon and is in fact the actual collision point (Figure 5). If the point is not inside the polygon, then

Possible Collision Point

Possible Collision Point

U' Axis

U' Axis

Odd number of intersections Inside Polygon

Even number of Intersections, Outside

Figure 5. Intersection Check

another plane must be intersected also and the same checks must be run for the remaining polygons. The planes are computed based upon the first three vertices of the individual polygons. Three points are all that are needed to compute a distinct plane. The remainder, if any, of the vertices are used in the algorithm to determine the edges of the polygon and the number of crossings of the polygon edges for the second ray shot from the saved intersection point. The plane computed from the first three vertices of the polygon along with the saved intersection point and the vertices of the polygon are all that are required to compute the number of crossings of the polygon edges. The algorithm is efficient since all edges are either rejected with no intersection at all or accepted as intersected. The algorithm also avoids the problem of points that lie exactly on the edge by placing those points either inside the checked polygon or outside it. The algorithm is as shown in Figure 6, using the following list of terms:

18

1. U' and V' - Perpendicular Coordinate axes for the plane of the polygon.
2. n - the variable number of the vertex from 0 to NV-1.
3. NV - Number of Vertices that comprise the current polygon.
4. SH, NSH - Sign Holder, Next Sign Holder.
5. a, b - Ordered variable vertices, a = 0 to NV-1 and b = (a +1) mod NV.
6. NC - Number of Crossings.

For the Number (NV) of vertices [Xn Yn Zn], where n = 0 to NV-1, project these   onto the dominant coordinate's plane, creating a list of vertices (Un, Vn).

Translate the (U,V) polygon so that the intersection point is the origin. Call these points (U'n, V'n).

Set the number of crossings, NC, to zero.

Set the sign holder, SH, as a function of V'0, the V' value of the first vertex of the first edge:{
Set to -1 if V'0 is negative
Set to +1 if V'0 is positive.
}
For each edge of the polygon formed by points (U'a, V'a) and (U'b,V'b), where a = 0 to NV -1, b= (a+1) mod NV:{

Set the next sign holder, NSH:{
Set to -1 if V'b is negative.
Set to +1 if V'b is positive.
}
If U'a is positive and U'b is positive then the line must cross +U', so increment NC.

Else if either U'a is positive or U'b is positive then the line might cross, so compute intersection on U' axis:{
if U'a - V'a*(U'b - U'a)/(V'b - V'a) > 0 then
the line must cross +U', increment NC.
}
Set SH = NSH.

Next edge.

}
If NC is odd, the point is inside the polygon, else it is outside.[1]

Figure 6. Haines' Algorithm

The key to making the algorithm work is to determine the dominant coordinate, and to then work only in that coordinate's plane. This simplifies the process enormously

---

1. Glassner, *An Introduction to Ray Tracing*, p. 56.

and allows for a much faster implementation in only two dimensions. Once that plane is found, the program shoots a ray from the intersection point along the positive U' axis and counts the number of edge crossings. This process is done for each edge of the polygon and once all edges have been checked for an intersection by the ray, the total is tested to determine if it is odd or even. This collision point, since there must be one, is then used to determine the proper response to the transient event.

3. Reactions

The proper response is performed by comparing the two objects involved in the collision to each other. The characteristics of each are compared to the other. The first check is to determine whether or not the objects involved are fixed or mobile. Once that is known the proper reaction can be displayed. A few general guidelines are applied to all collisions. The larger massed object inflicts more damage on the smaller massed object. The fixed object has no ability to shift away from the point of contact and consequently suffers more damage than a mobile object with the ability to spring away (Figure 7). A large fixed object, such as a bunker or large rock, can withstand a much larger force of impact than a small fixed object. A large fixed object also inflicts much more damage to the mobile object that struck it. A small mobile object suffers damage if it is hit by a large mobile object at angles that are near multiples of 90 degrees. At smaller angles, even small vehicles are able to bounce away from the impact with a minimum of damage. Consequently, the collision response is limited to a few instances. For fixed objects, the responses include several degrees of damage, based upon the speed and mass of the colliding object. Up to three levels of damage plus the original undamaged fixed object are available for display after a collision. For mobile objects, the response depends upon the angle of impact as well as the speed and mass of the two involved objects. The mobile object reacts by either bouncing away or being destroyed and exploding. In the special case of contact by munitions, the only response is an explosion. The constants are checked in order of size, i.e., constant1 > constant2 > constant3 > constant4. The algorithm is as shown in Figure 8.

Old
Azimuths

OBJ
B

OBJ
A

Figure 7a. Mobile Object Contact

Spring
Force

New
Azimuths

OBJ
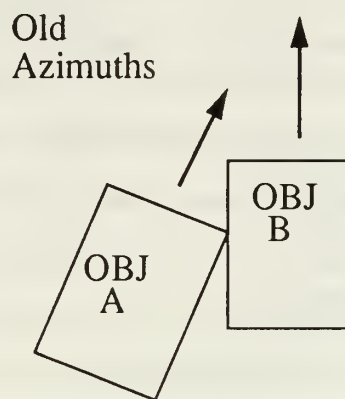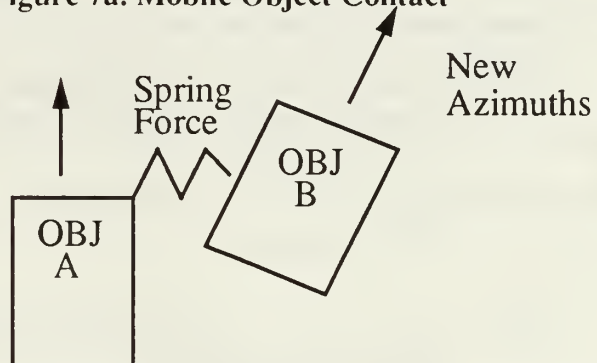B

OBJ
A

Figure 7b. Reaction

```
If fixed object{
        if mass of colliding vehicle >= constant1 then
                damage level == HI;
        else if mass of colliding vehicle < constant1 && >= constant2 then
                damage level == MED;
        else if mass of colliding vehicle < constant2 && >= constant3 then
                damage level == LOW;
        else if mass of colliding vehicle <= constant3 then
                damage level == NONE;
}
Else if mobile object and other object is fixed then{
        Check for mass of fixed object;
        if large and mobile object is moving quickly
                Kill mobile object;
        if small or moving slowly{
                continue on course;
                damage fixed object;
                decrease speed of mobile object;
        }
}
Else if mobile object and other object is also mobile{
        Check for angle of impact;
        If ((angle >=85 && <= 95) || (angle >= 175 && angle <= 185) ||
            (angle <= 5) || (angle >= 355) || (angle >= 265 && angle <= 275)){
                if (speed of colliding vehicle > constant){
                        Kill both;
                }
                else
                        Stop both;
        }
        Else {
                Bounce off each other;
                Diminish speed of both;
        }
}
```

Figure 8. Collision Response Algorithm

The limited number of options available for the response to the collision keep the response fast to maintain the real-time criteria. The collision point itself is saved to pass to another function along with the direction of travel and the object type. This function's implementation can be seen in [Mona91]. That work uses all of the physical characteristics of the object to create the more accurate response that goes beyond the scope of this work.

IV. RESULTS

A. PERFORMANCE

The performance of the overall system is not affected by the addition of the collision detection and response modules. The response time for detection is adequate for fixed objects regardless of the speed of the moving objects. However, for collisions between two high speed objects, collision detection is sometimes too slow. When vehicles are traveling too quickly, i.e., faster than about 216 KPH each, they pass through each other rather than colliding. For example: two tanks, each with a radius of three meters, would have to travel six meters in one frame to do this. This is a totally unrealistic speed for a ground or water vehicle but not for an air vehicle as shown in Table 2. This is due to the inability of the functions to calculate the positions of both vehicles quickly enough to realize that a collision was supposed to have occurred. A time interpolation of the movement of the vehicles would solve this. At speeds below 216 KPH, the detection functions between two moving objects work.

B. ACHIEVEMENT OF GOALS

The collision detection and response is done in real-time. Real-time is defined as about ten frames per second. This is the speed of NPSNET-1, and the speed of NPSNET-2 is not degraded by the addition of the detection and response modules.

NPSNET-2 does not detect all collisions; however, it detects the vast majority. All collisions between fixed and moving objects are detected and give a response that is done in real-time and in a realistic manner. The collisions between moving objects is probably adequate due to the normally slow speed of tactical vehicles. The effects are realistic and are close to what would occur in the real world if the collisions actually occurred.

The system is compatible with future upgrades of hardware assuming those upgrades make use of the GL libraries that are currently used. If those libraries change then the functions contained within them should also change and remain compatible with current hardware.

V. CONCLUSIONS

A. MERITS OF THE WORK

NPSNET-2 is compatible with SIMNET which was one of the original goals. Realism for simulation is maintained by allowing correct physically-based modeling characteristics to occur in response to transient events within the virtual world. The real-time collision detection and response allow the user to interact with the virtual world and with other networked players. The speed of the entire system is not affected by the addition of the collision detection and response.

B. ASSUMPTIONS AND LIMITATIONS

One assumption was made to speed up and enhance the program. It is that only two objects or vehicles are involved in any one collision or explosion. The assumption is reasonable since collisions rarely occur between three or more objects.

Getting an algorithm fast enough to handle real-time collision detection and response was difficult. The collision detection and response for fixed objects was simpler since there were fewer fixed objects and their position remained a constant. However, the moving objects were much more difficult to track for collisions since they constantly moved and there were many more vehicles in the virtual world than the average number of fixed objects per gridsquare.

One of the system's limitations is that it cannot detect collisions between two quickly moving objects as stated in Chapter IV. Time interpolation needs to be integrated into the system in order to detect all collisions between two moving objects. Moreover, the system will only detect collisions between two objects, whether they are moving or not. Therefore, minor limitation is that collisions between three or more objects are not checked for

simultaneously. An implied limitation is that all objects in the world are spherical, when in fact, few are. Detections are done on spheres and therefore have a margin for error on the virtual objects. The final limitation is that fixed objects that are large, in comparison to the size of the gridsquare, and are located near the borders of gridsquares may not be detected until after they have been penetrated (Figure 9). However, there are only a small number of these objects in the virtual world.

Vehicle

Object Located in this Gridsquare

Azimuth

125 Meters

Figure 9. Border Object Limitation

C. IDEAS FOR FUTURE PROJECTS

Only a few of the physically-based modeling characteristics were used in determining the response to collisions. Obviously, the remainder of those characteristics can be added for more realism. Actual physics laws were also avoided due to their computational intensity. However, for every area that is added to obtain more realistic affects there is a cost in time. Too many will cause the real-time constraints to be exceeded and therefore cost more than the system can afford. Faster hardware and/or software will allow these constraints to be met. Future work is needed to include all of the physically-based

characteristics. Additionally, a time interpolation of vehicle movement is needed in order to prevent vehicles from passing through one another.

VI. APPENDIX A

This appendix contains snapshots of the screen from the virtual Fort Hunter-Liggett, including collision responses for several cases.
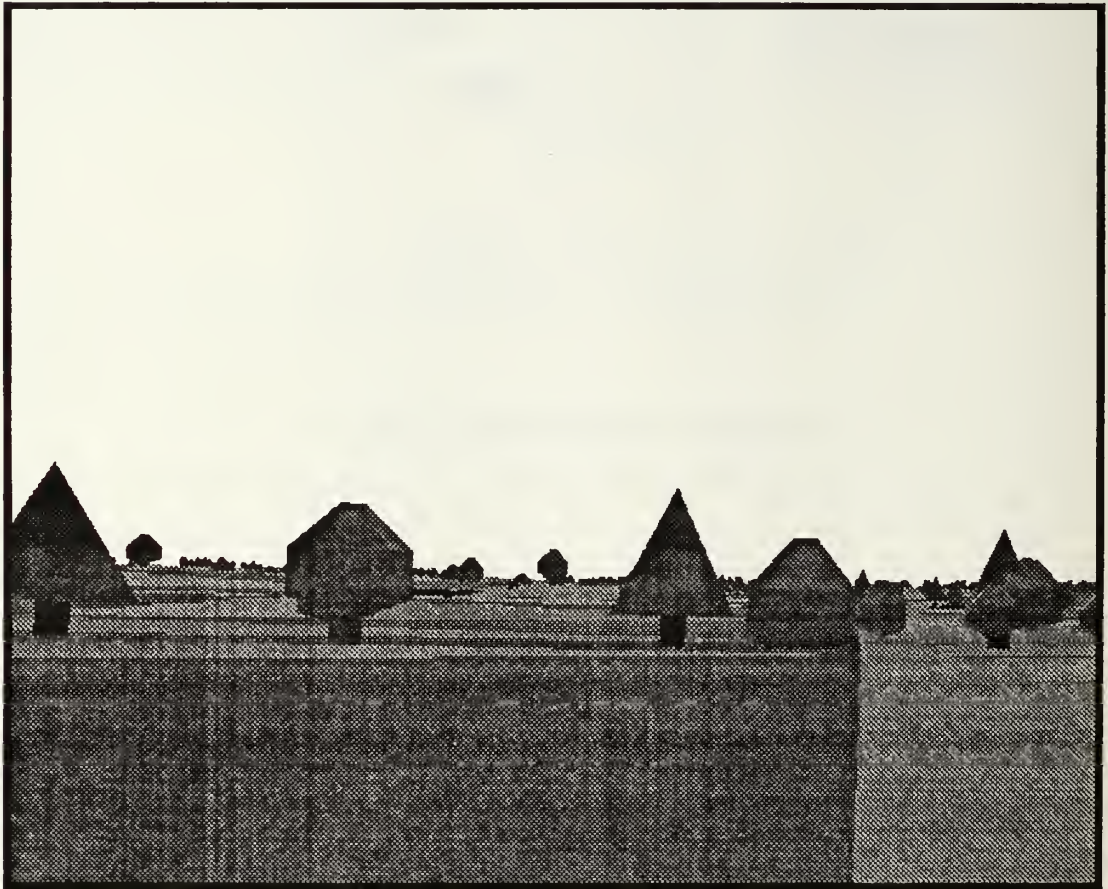

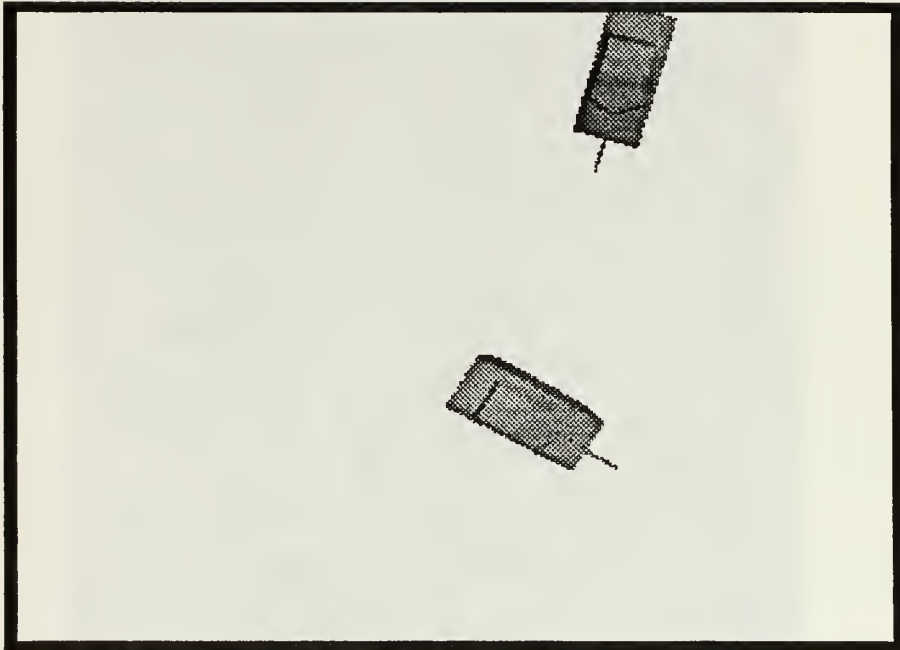**Figure A.1-A Typical Scene From Inside a Vehicle in the Virtual Hunter-Liggett**

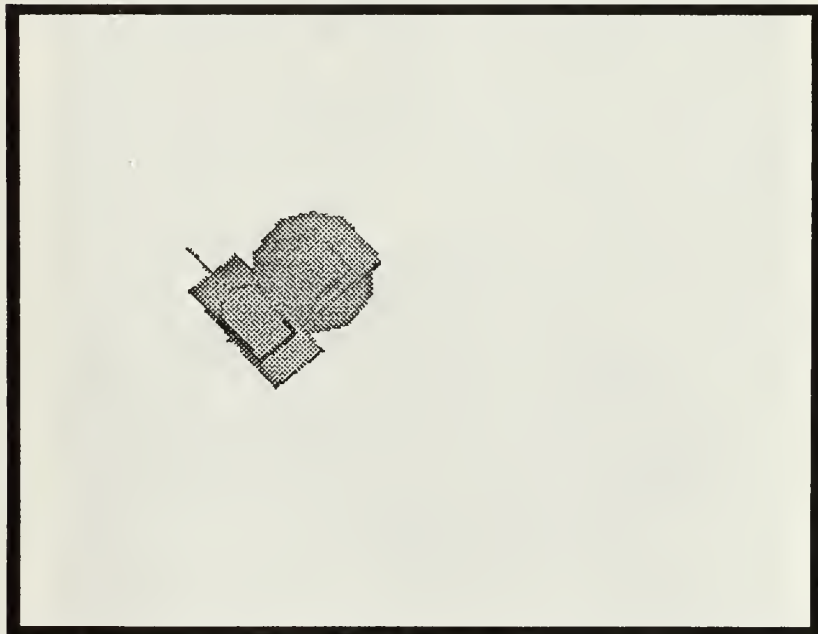Figure A.2-Imminent Collision Between Two Vehicles

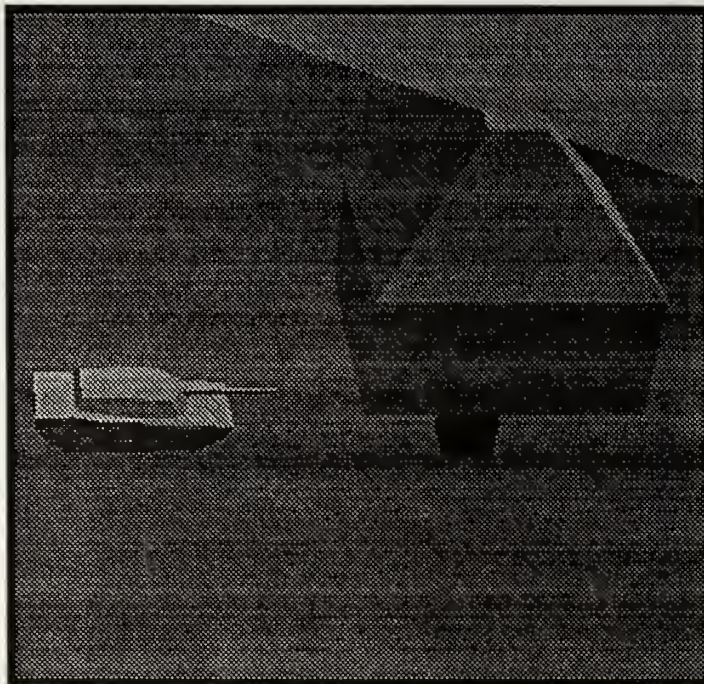Figure A.3-Broadside Collision Result, Explosion and Destruction
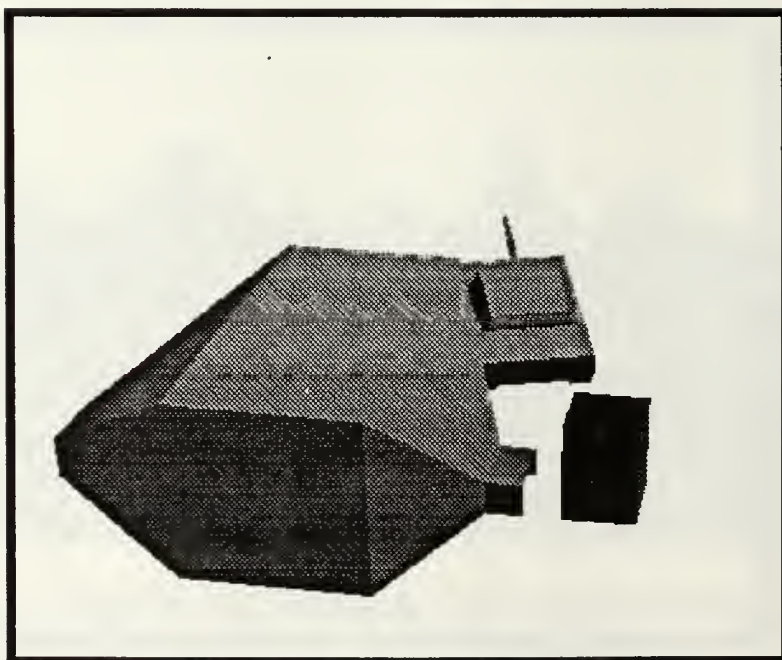
Figure A.4-Imminent Collision with Tree

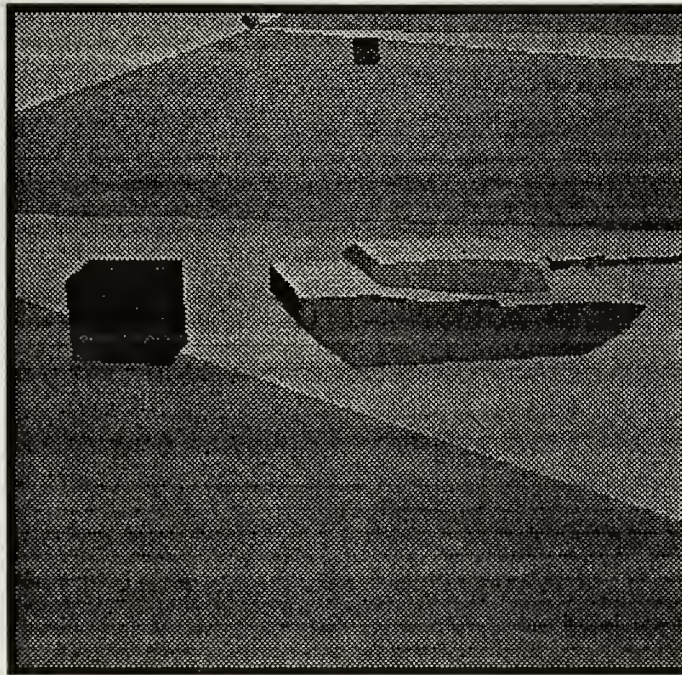**Figure A.5-Slow Speed Collision with a Tree**

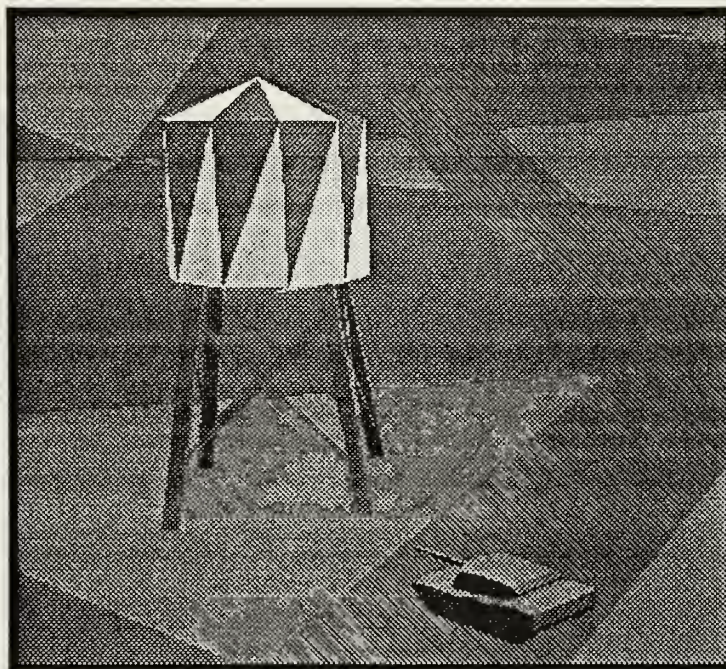Figure A.6-High Speed Collision with a Tree

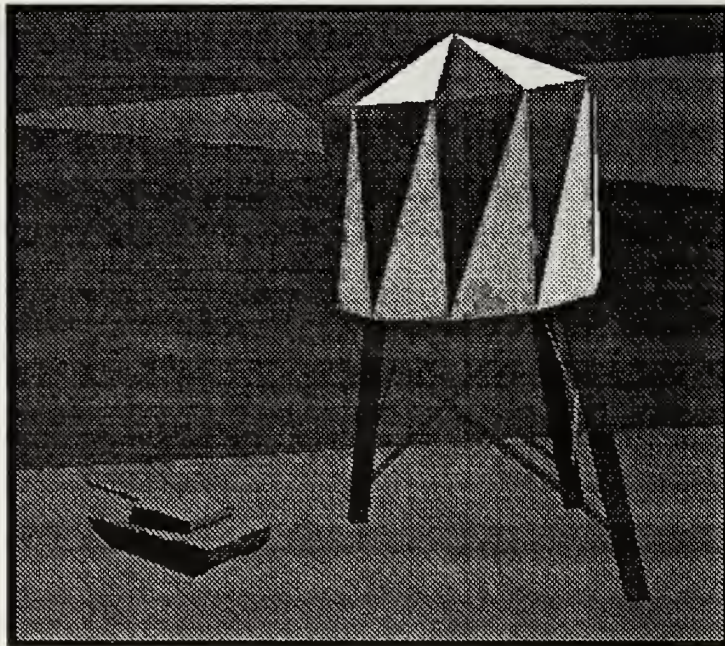Figure A.7-Imminent Collision with Tower
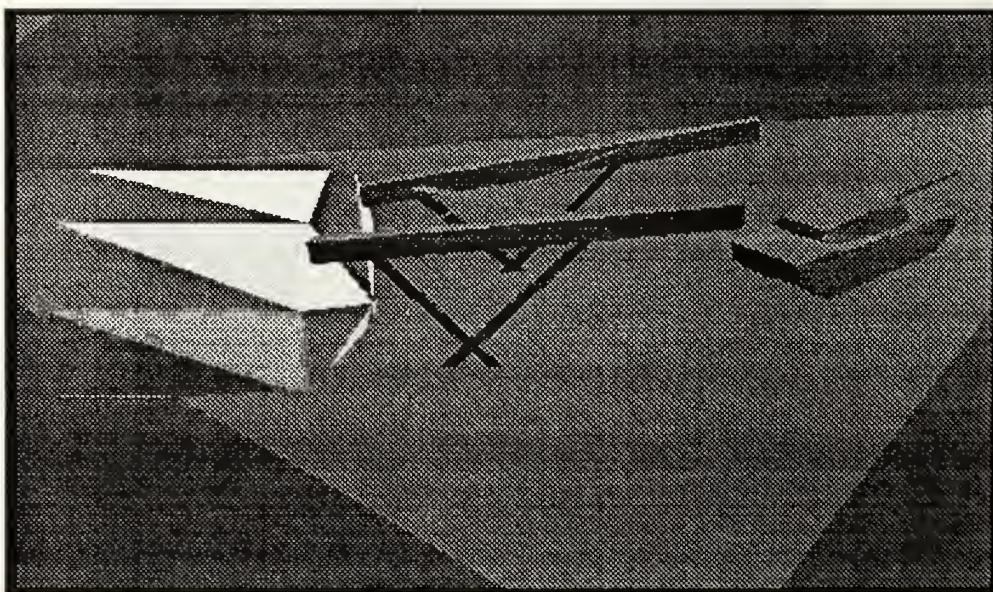
Figure A.8-Slow Speed Collision with Tower

Figure A.9-High Speed Collision with Tower

REFERENCES

[Chee90]   Cheeseman, Curtis P., *Moving Platform Simulator III: An Enhanced High-Performance Real-Time Graphics Simulator With Multiple Resolution Display and Lighting*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, June 1990.

[Garv88]   Garvey, Richard E., Jr., and Monday, Paul, "SIMNET (SIMulator NETworking)", BBN Systems and Technologies, Bellevue, WA, July 28, 1988.

[Glas89]   Glassner, Andrew S., editor, *An Introduction to Ray Tracing*, Academic Press, San Diego, CA, 1989.

[Glas90]   Glassner, Andrew S., editor, *Graphics Gems*, Academic Press, San Diego, CA, 1990.

[Hahn88]   Hahn, James K., "Realistic Animation of Rigid Bodies", *Computer Graphics*, Vol 22, no. 4, pp. 299 - 308, August 1988.

[Hopc83]   Hopcroft, J. E., Schwartz, J. T. and Sharir, M., "Efficient Detection of Intersections Among Spheres", *The International Journal of Robotics*, Vol 2, no. 4, pp. 77 - 80, Winter 1983.

[Kern78]   Kernighan, Brian W., and Ritchie, Dennis M., *The C Programming Language*, Prentice-Hall, Englewood Cliffs, NJ, 1978.

[Lang90]   Lang, Eric and Wever, Pete, *SDIS Version 3.0 User's Guide: Interchange Specification, Class Definitions, Application Programmer's Interface*, BBN Systems and Technologies, Bellevue, WA, August 1990.

[Mona91]   Monahan, James G., *NPSNET: Physically-Based Modeling Enhancements to an Object File Format*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, September 1991.

[Moor88]   Moore, Mathew and Wilhelms, Jane, "Collision Detection and Response for Computer Animation", *Computer Graphics*, Vol 22, no. 4, pp. 289 - 298, August 1988.

[Terz87]   Terzopoulos, Demetri, Platt, John, Barr, Alan and Fleisher, Kurt, "Elastically Deformable Models", *Computer Graphics*, Vol 21, no. 4, pp. 269 - 278, July 1987.

[Uchi83]   Uchiki, Tetsuya, Ohashi, Toshiaki and Tokoro, Mario, "Collision Detection in Motion Simulation", *Computers and Graphics*, Vol 7, no. 3-4, pp. 285 - 293, 1983.

[Winn89]   Winn, Michael C., and Strong, Randolph P., *Moving Platform Simulator II: A Networked Real- Time Simulator with Intervisibility Displays*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, June 1989.

[Zyda91]   Zyda, Michael J., and Pratt, David R., "NPSNET: A 3D Visual Simulator for Virtual World Exploration and Experimentation", *1991 SID International Symposium, Digest of Technical Papers*, Society for Information Display, Playa Del Rey, CA, pp. 361 - 363, May 1991.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center   2
Cameron Station
Alexandria, VA   22304-6145

Dudley Knox Library   2
Code 52
Naval Postgraduate School
Monterey, CA   93943-5002

Chairman, Code CS   2
Computer Science Department
Naval Postgraduate School
Monterey, CA   93943-5100

Dr. Michael J. Zyda   6
Naval Postgraduate School
Code CS, Department of Computer Science
Monterey, CA   93943-5100

David M. Pratt   2
Naval Postgraduate School
Code CS, Department of Computer Science
Monterey, CA   93943-5100

Captain William D. Osborne   1
3012 Gentry Road
Virginia Beach, VA 23452